SOCIS16 at GNU OCTAVE

Improve Iterative Methods for Sparse Linear Systems

by

Cristiano Dorigo

March 21, 2017

Introduction

The general idea of an iterative methods for a linear system $Ax = b$, is to obtain a sequence of approximations for the problem using each time the previous iteration.

In particular the methods which I worked on are:

Improved Methods PCG, GMRES, BICG, BICGSTAB, CGS;

New Method TFQMR.

Table of Contents

This presentation will be split in three parts:

PART ONE: Preliminar study and private functions;

PART TWO: Main improvements common to all methods;

PART THREE: Particular cases.

PART ONE

PART ONE

In this first part we will see how to consider similar part of codes in all the methods and a brief study of how to make the methods more efficient.

Check the input

All of the iterative methods in Octave have a similar input/output sintax, for example:

$$[x, \mathit{flag}, \mathit{relres}, \mathit{iter}, \mathit{resvec}] = \mathrm{bicg}(A, b, \mathit{tol}, \mathit{maxit}, M_1, M_2, x_0)$$

Not all of them are necessary to use the method: If some of the inputs are not passed, they are used default values.

Each iterative method has an initial similar block of codes to check the inputs and fill the empty ones. To avoid this:

___default___input___.*m*

A, M_1, M_2 matrices or function?

The inputs A, M_1, M_2 are in general matrices, but they can be passed also as functions handle.

How can we face these different situations?

Two strategies in the implemented codes:

Algorithm 1 Strategy 1

1: **if** A matrix **then**
2: $\quad w = A * v$
3: **else if** A function handle **then**
4: $\quad w = \text{feval}(A, v)$
5: **end if**

Algorithm 2 Strategy 2

1: **if** A matrix **then**
2: $\quad Ax = @(x)A * x$
3: **else if** A function handle **then**
4: $\quad Ax = A$
5: **end if**
6: $w = Ax(v)$

A, M_1, M_2 matrices or function? - pt 2

Which one is more efficient?

Table An example of the times for each strategy with matrices of different sizes

Size	Str. 1	Str. 2 (A matrix)	Str. 2 (A function)
100	0.0040579	0.0023341	0.0049782
200	0.0088201	0.0072379	0.010650
500	0.016917	0.013485	0.015626
1000	0.37738	0.38262	0.37756
2000	1.0484	0.97292	0.97577

A, M_1, M_2 matrices or function? - pt.

Then for each method two subscripts:

- When A (and, if passed to the method) M_1, M_2 are matrices;
- When at least one of A, M_1 or M_2 is a function handle. All the others are converted to functions.

The private function

___checkandstring___.*m*

to check the type of A, M_1 and M_2

PART TWO

PART TWO

In this part we will see the main improvements applied, common to all the methods.

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

FLAG $= 0$: when the algorithm converges;

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

- FLAG $= 0$: when the algorithm converges;
- FLAG $= 1$: when the algorithm reachs the maximum number of iterations without convergence;

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

- FLAG $= 0$: when the algorithm converges;
- FLAG $= 1$: when the algorithm reachs the maximum number of iterations without convergence;
- FLAG $= 2$: when the preconditioner matrix $M = M_1 * M_2$ is ill-conditioned;

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

- FLAG $= 0$: when the algorithm converges;
- FLAG $= 1$: when the algorithm reachs the maximum number of iterations without convergence;
- FLAG $= 2$: when the preconditioner matrix $M = M_1 * M_2$ is ill-conditioned;
- FLAG $= 3$: in case of stagnation;

FLAG output

The most important improvement of the methods is the FLAG output. Now is more Matlab compatible.

In general in Matlab the FLAG variable can have one of the following values:

- FLAG $= 0$: when the algorithm converges;
- FLAG $= 1$: when the algorithm reachs the maximum number of iterations without convergence;
- FLAG $= 2$: when the preconditioner matrix $M = M_1 * M_2$ is ill-conditioned;
- FLAG $= 3$: in case of stagnation;
- FLAG $= 4$: when one of the scalar quantities calculated during the algorithm became too small or too large to continue computing.

FLAG = 2

FLAG $= 2$ in Matlab means that the preconditioner matrix $M = M_1 * M_2$ is ill-conditioned and when detected it, the algorithm stops.

A non-singular matrix A is *ill-conditioned* if its condition number (which is $\frac{\sigma_{max}}{\sigma_{min}}$, where $\sigma_{max}, \sigma_{min}$ are respectively the maximum and the minimum singular value of A) is high.

Strange fact: in Matlab, the only cases in which I obtained this FLAG were when M was singular or had some NaN as coefficients.

How to detect M singular?

Try-catch approach, when in the algorithm M is used the first time.

```
1: try
2:     warning ("error","Octave:singular-matrix","local");
3:     w = M\v;
4: catch
5:     flag = 2;
6:     break;
7: end_try_catch
```

FLAG = 3

In Matlab this flag appears when the algorithm stagnates. In its help is written "two consecutives iterates were the same".

This string is not very clear and seems impossible to get two identical consecutives iterates.

Following criteria for the stagnation:

$$\text{norm}(x - x_{pr}) \leq \text{norm}(x) * \text{eps}$$

where x_{pr} is the previous iterates.

Other Improvements

Briefly, other small improvements are:

Other Improvements

Briefly, other small improvements are:

- As in Matlab, the iterates returned is the one already computed that makes the residual smaller. Also, the output variable ITER shows the iteration when X is computed;

Other Improvements

Briefly, other small improvements are:

- As in Matlab, the iterates returned is the one already computed that makes the residual smaller. Also, the output variable ITER shows the iteration when X is computed;
- Added in each method some output strings that comes when are requested less than two outputs. These strings are are printed in substitution of the flag output;

- The algorithms now works also with complex arguments, and not only with real ones;

- The algorithms now works also with complex arguments, and not only with real ones;
- Improved the documentations, in particular in the part regarding the preconditioners (now are clearly described the correct preconditioner procedure applied to each method);

- The algorithms now works also with complex arguments, and not only with real ones;
- Improved the documentations, in particular in the part regarding the preconditioners (now are clearly described the correct preconditioner procedure applied to each method);
- Added demos and tests, in particular to check the behaviour of the subscripts, the flag output and the correctness with complex inputs.

PART THREE

PART THREE

In this last part we consider some particular cases that I faced with the improvements of PCG, GMRES and BICG.

PCG and Complex Numbers

The PCG algorithm, in general, works when the input matrix A is *Hermitian Positive Definite* (HPD), that is when it holds:

- $A = A^H$
- $x^H A x > 0$ $(= 0$ when $x = 0)$ for any vector x

At each PCG-iteration, the approximant is obtained in the following way:

$$x_m = x_{m-1} + \alpha p_{m-1}$$

where $\alpha = \frac{\|r_{m-1}\|_2^2}{p_{m-1}^H A p_{m-1}}$ and $r_{m-1} = b - A x_{m-1}$.

Thus α must be positive. If, by computation, we have $\alpha \leq 0$ then the algorithm can't continue and it must be stopped.

PCG and Complex Numbers - part 2

In the Octave PCG, there is the following simple check:

```
1: if  α ≤ 0 then
2:     flag = 4;
3:     break;
4: end if
```

In Octave, if we try to compare two complex numbers, it compares their modules.

Problem: if α is complex, in Octave is always positive!

And in Matlab?

In Matlab if we compare two complex numbers, it checks only the real part of the arguments.
What is the behaviour of the Matlab PCG with complex inputs?

If the matrix A has symmetric real part (that is $\Re(A) = \Re(A^T)$) and any imaginary part, then the Matlab-PCG doesn't notice that the matrix is not-Hermitian.

New control for α

Now the numerator and the denominator of α are checked separately.
Suppose that δ is the numerator of α then the check is:

1: **if** $(\Re(\delta) \leq 0)$ or $(|\Im(\delta)| \geq |\Re(\delta)| * tol)$ **then**
2: flag $= 4$;
3: break;
4: **end if**

GMRES and Inner/Outer Iterations

The GMRES algorithm input variable more: RESTART.
When this parameter is passed, the algorithm performs a number of iteration equal to RESTART, then it sets $x_0 = x_{RESTART}$ and starts again from the beginning with x_0 as initial guess.

GMRES and Inner/Outer Iterations

The GMRES algorithm input variable more: RESTART.
When this parameter is passed, the algorithm performs a number of iteration equal to RESTART, then it sets $x_0 = x_{RESTART}$ and starts again from the beginning with x_0 as initial guess.

The total number of iterations are RESTART*MAXIT if RESTART is passed (in this case RESTART must be $\leq n$), and MAXIT if RESTART not passed (in this case MAXIT must be $\leq n$).

GMRES and Inner/Outer Iterations

The GMRES algorithm input variable more: RESTART.
When this parameter is passed, the algorithm performs a number of iteration equal to RESTART, then it sets $x_0 = x_{RESTART}$ and starts again from the beginning with x_0 as initial guess.

The total number of iterations are RESTART*MAXIT if RESTART is passed (in this case RESTART must be $\leq n$), and MAXIT if RESTART not passed (in this case MAXIT must be $\leq n$).

Then I fixed all the corner cases that can happen.

GMRES and Inner/Outer Iterations

Because of RESTART, the output variable ITER has a different meaning: it shows at which Outer (ITER(1)) and Inner (ITER(2)) Iteration the approximant x is computed.

GMRES and Inner/Outer Iterations

Because of RESTART, the output variable ITER has a different meaning: it shows at which Outer (ITER(1)) and Inner (ITER(2)) Iteration the approximant x is computed.

OUTER ITERATION: It is the number of times that the method restarts. It is $\leq MAXIT$;

GMRES and Inner/Outer Iterations

Because of RESTART, the output variable ITER has a different meaning: it shows at which Outer (ITER(1)) and Inner (ITER(2)) Iteration the approximant x is computed.

OUTER ITERATION: It is the number of times that the method restarts. It is $\leq MAXIT$;

INNER ITERATION: It is the number of iteration performed after the last restart. It is $\leq RESTART$.

GMRES and Inner/Outer Iterations

Because of RESTART, the output variable ITER has a different meaning: it shows at which Outer (ITER(1)) and Inner (ITER(2)) Iteration the approximant x is computed.

OUTER ITERATION: It is the number of times that the method restarts. It is $\leq$ *MAXIT* ;

INNER ITERATION: It is the number of iteration performed after the last restart. It is $\leq$ *RESTART*.

The approximant x is computed at the iteration $(ITER(1) - 1) * RESTART + ITER(2)$.

BICG, BICGSTAB and FLAG = 4

The general idea of the BICG algorithm is to solve at the same time the problem $Ax = b$ and the dual problem $A^H y = b$. Then at each iteration we have the approximants:

$$x_{m+1} = x_m + \alpha p_m, \qquad y_{m+1} = y_m + \alpha p_m^\star$$

where: $\alpha = \frac{(r_m, r_m^\star)}{(Ap_m, p_m^\star)}$ $\quad r_m = b - Ax_m$ $\quad r_m^\star = b - A^H y_m$.

If $\alpha = 0$, there are two possibilities:

- r_m or $r_m^\star$ is zero;
- r_m and $r_m^\star$ are orthogonal.

BICG, BICGSTAB and FLAG = 4 -

If we are in the first case with $r_m = 0$, we're happy because it means that x_m is correct.

If we are in the first case with $r_m^\star = 0$, or in the second case, then the algorithm can't continue, since the approximant $x_{m+1} = x_m$. Moreover the quantity $(r_m, r_m^\star)$ brings to a division by zero.

BICG, BICGSTAB and FLAG = 4 -

If we are in the first case with $r_m = 0$, we're happy because it means that x_m is correct.

If we are in the first case with $r_m^\star = 0$, or in the second case, then the algorithm can't continue, since the approximant $x_{m+1} = x_m$. Moreover the quantity $(r_m, r_m^\star)$ brings to a division by zero.

In Matlab is not clear what does it means FLAG $= 4$ for BICG and BICGSTAB. I think that this FLAG is linked to this fact, but unfortunately it's only a supposition.

THE END

Thanks too..

- The European Space Agency for giving me the possibility of taking part to the Summer of Code in the Space program
- The mentors Marco Caliari and Carlo de Falco for their support and suggestions;
- GNU Octave and its community.